Лекция 6. Наследование

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Объектно-ориентированное программирование

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель лекции

Цель лекции

Изучить механизм наследования в C++: отношения базового и производного классов, доступ к членам базового класса и особенности работы с указателями на объекты.

Ожидаемые результаты

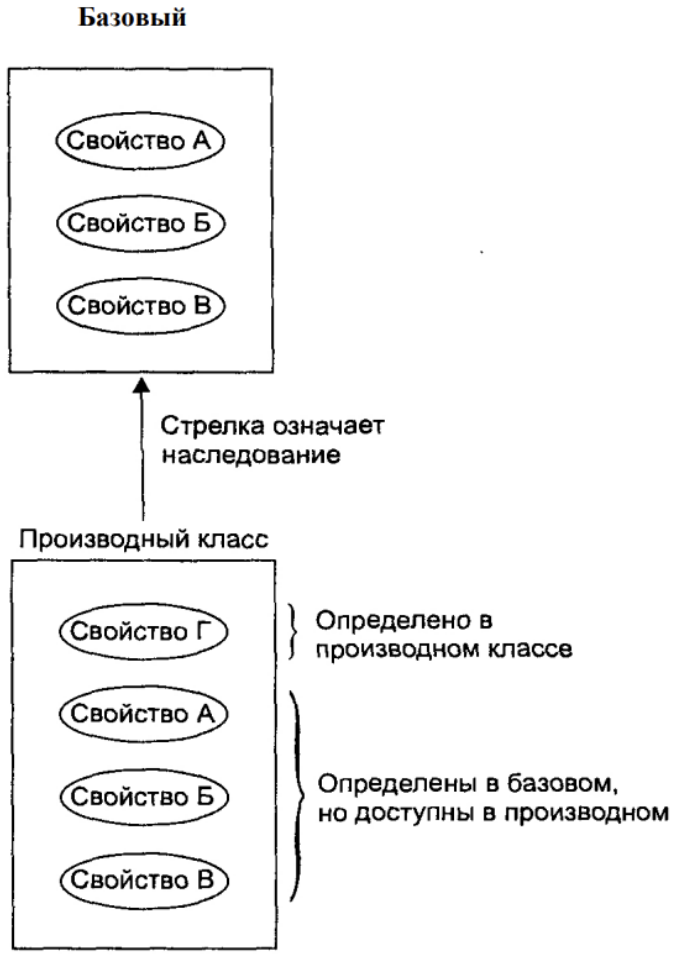
1. Определять производные классы
2. Объяснять, какие члены базового класса доступны в производном
3. Использовать повторно код ранее созданных классов
4. Работать с указателями на объекты базового и производного классов

План лекции

- Наследование
- Преимущество
- Базовый и производный класс
- Примеры
- Объекты базовых и производных классов
- Доступ к членам базового и производного класса
- Указатели

Наследование

- Наиболее значимой после классов возможностью ООП является наследование. Это процесс создания новых классов, называемых наследниками или производными классами, из уже существующих или базовых классов. Производный класс получает все возможности базового класса, но может также быть усовершенствован за счет добавления собственных. Базовый класс при этом остается неизменным.



Преимущество

- Наследование — важная часть ООП. Выигрыш от него состоит в том, что наследование позволяет использовать существующий код несколько раз. Имея написанный и отлаженный базовый класс, мы можем его больше не модифицировать, при этом механизм наследования позволит нам приспособить его для работы в различных ситуациях. Используя уже написанный код, мы экономим время и деньги, а также увеличиваем надежность программы. Наследование может помочь и при начальной постановке задачи программирования, разработке общей структуры программы.
- Важным результатом повторного использования кода является упрощение распространения библиотек классов. Программист может использовать классы, созданные кем-то другим, без модификации кода, просто создавая производные классы, подходящие для частной ситуации.

Базовый и производный класс

- Наследование влияет на размер и поведение объектов производного класса двумя способами. Очевидно, что объект производного класса будет содержать все элементы данных производного класса. Однако он также будет содержать элементы данных базового класса. Таким образом, объект производный класс всегда будет больше, чем объект базового класса. (Единственным исключением является случай, когда ни базовый класс, ни производный класс не имеют данных-членов. В этом случае объекты как базового, так и производного класса занимают по одному байту каждый.)
- Очевидно, по отношению к объекту класса производного класса, мы можем вызывать общедоступные функции-члены производного класса в любой глобальной функции, не являющейся членом. Однако мы также можем вызывать общедоступные функции-члены базового класса. (Есть исключения из этого. Обстоятельства при которых возникают эти исключения, описаны далее в этой главе.)

Пример наследования

```
class A
{
int x;
public:
void setX(const int = 0);
int getX()const;
};

class B : public A //inheriting from A
{
int y;
public:
void setY(const int = 0);
int getY()const;
};
```

```
void A::setX(const int pX)
{
x = pX;
}
int A::getX() const
{
return x;
}
 void B::setY(const int pY)
 {
 y = pY;
 }
 int B::getY() const
 {
 return y;
 }
```

Наследования (main funtion)

```
int main() {
cout << sizeof(A) << endl << sizeof(B) << endl;
B B1; //an object of the derived class
B1.setX(1); //OK: calling a base class member function with respect to a derived class object
B1.setY(3);
cout << B1.getX() << endl; //OK: calling a base class
//member function with respect to
//a derived class object
cout << B1.getY() << endl;
return 0;
}
```

Output

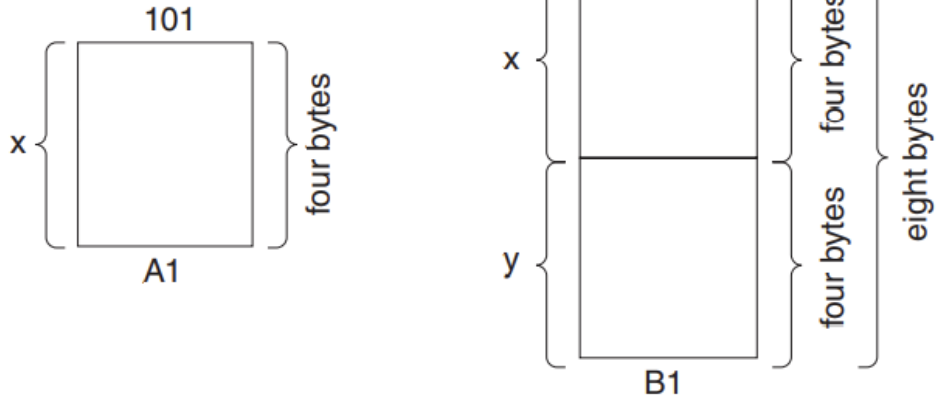
4
8
1
3

Использование в практике

- На практике программист библиотеки определяет определенный класс и его функции-члены. Другой заинтересованный программист, чтобы создать свое приложение, затем наследуется от этого класса и добавляет только специальные элементы данных и код для обработки этих дополнительных элементов данных в производном классе.

Объекты базового и производного класса

- Теперь многие изучающие C++ могут начать верить, что объекты производного класса наследуются от объектов базового класса. Это неверно, так как объект производного класса никак не связан с другим одновременно существующим объектом базового класса.
- Объект класса A (скажем, «A1») будет занимать четыре байта, содержащих только «x». В то время как объект класса B (скажем, «B1») будет занимать другой блок из восьми байтов, содержащий как «x», так и «y», как показано на рисунке




```
class A
{
int x;
public:
void setX(const int = 0);
int getX()const;
};

class B : public A //inheriting from A
{
int y;
public:
void setY(const int = 0);
int getY()const;
};
```

Доступ к членам базового класса в производном классе

- В функциях производного класса можно получить доступ только к публичным членам базового класса (также можно получить доступ к защищенным членам базового класса; защищенные члены мы обсудим позже).Но к закрытым членам базового класса нельзя получить доступ.Предположим, в функции B::setY() мы пишем

```
x=y;
```

- компилятор сообщит об ошибке о том, что закрытые члены базового класса недоступны. (В данном случае мы пытаемся получить доступ к «x» в функции-члене производного класса. Но «x» является закрытым членом базового класса.

- Но мы можем получить доступ к функциям A::setX() и A::getX() в функциях-членах производного класса, поскольку они являются открытыми членами базового класса. Закрытые члены базового класса остаются закрытыми по отношению к функциям-членам производного класса. Следующее строки кода демонстрируют это:

```
void B::setY(const int q)
{
y=q;
setX(y); //x=y
}
```

Доступ к членам при наследовании

Член базового класса	public-наследование	protected-наследование	private-наследование
public	public	protected	private
protected	protected	protected	private
private	недоступен	недоступен	недоступен

Закрытые (private) члены базового класса недоступны в производном классе при любом типе наследования. Для доступа к ним используются открытые или защищённые (protected) методы.

Дружественный класс

```
class B;

class A
{
          friend class B;
          int x;
};

class B
{
          void fB(A * p)
          {
            p->x=0; //OK: B is a friend of A
          }
};

class C : public B
{
          void fC(A * p)
          {
           p->x=0; //ERROR: C is not a friend of A
          //despite being derived from a friend
          }
};
```

- Дружба не передается по наследству. Класс не становится другом для класса, для которого является другом его родитель. След код иллюстрирует это.

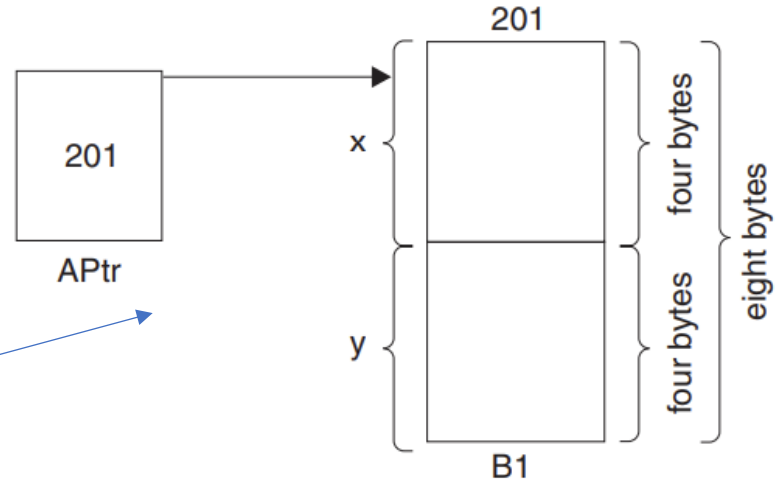
Указатели базового и производного класса

Указатель базового класса, указывающий на объект производного класса

```
/*Beginning of A.h*/
class A
{
   public:
      int x;
};
/*End of A.h*/

/*Beginning of B.h*/
#include“A.h”
class B : public A //derived from A
{
   public:
      int y;
};

 #include“B.h”          //from listing 5.03
 void main()
 {
    A * APtr;
    B B1;
    APtr=&B1;          //line 1 – OK: base class pointer points at
                       //derived class’s object
    APtr->x=10;        //line 2 – OK: accessing base class member
                       //through base class pointer
    APtr->y=20;        //line 3 - ERROR: y not found in class A
 }
```



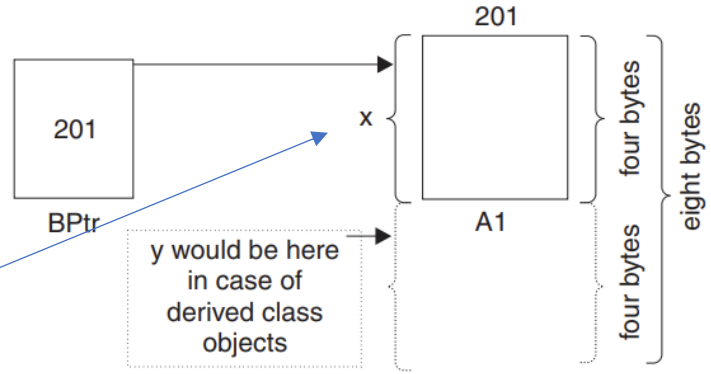
Указатели базового и производного класса

Указатель производного класса, указывающий на объект базового класса

```
/*Beginning of A.h*/
class A
{
   public:
      int x;
};
/*End of A.h*/

/*Beginning of B.h*/
#include“A.h”
class B : public A //derived from A
{
   public:
      int y;
};
```

```
#include“B.h”          //from listing 5.03
void main()
{
   A A1;
   B * BPtr;
   BPtr=&A1;          //line 1 – ERROR. Cannot convert from B* to
                      //A*.
   BPtr->x=10;        //line 2 – OK. Derived class pointer
                      //accesses base class member.
   BPtr->y=20;        //line 3 – OK. Derived class pointer
                      //accesses derived class member.
}
```



Line1 не будет скомпилирована, потому что скомпилируется строка 3. Указатель производного класса не может указывать на объект базового класса. Давайте посмотрим, почему. «BPtr» относится к типу «B *». Предполагается, что он указывает на объекты производного класса B. Следовательно, он также может обращаться к «y». «BPtr» указывает на«A1», который занимает всего четыре байта. Однако он также может получить доступ к значению, содержащемуся в следующих четырех байтах. В классе B есть член с именем «y». Таким образом, «BPtr» может получить доступ к области памяти, которая не была выделена. Следовательно, указатель типа производного класса не может безопасно указывать на объект базового класса, как показано на рис. 5.4.

Выводы

1. Наследование — создание нового (производного) класса на основе существующего (базового) с повторным использованием его кода.
2. Объект производного класса содержит поля базового класса и собственные поля, поэтому занимает больше памяти.
3. Закрытые (private) члены базового класса недоступны в производном; к ним обращаются через открытые методы.
4. Тип наследования (public, protected, private) определяет уровень доступа к унаследованным членам.
5. Указатель на базовый класс может ссылаться на объект производного класса.

Контрольные вопросы

1. Что такое наследование и какие преимущества оно даёт?
2. Что такое базовый и производный класс? Как записывается наследование в C++?
3. Чем объект производного класса отличается от объекта базового по составу и размеру?
4. Какие члены базового класса доступны в производном классе?
5. Как тип наследования влияет на доступ к унаследованным членам?
6. Какие указатели можно присваивать друг другу: на базовый и производный классы?

Список литературы

Основная литература

1. Лафоре Р. Объектно-ориентированное программирование в C++. 4-е изд. — СПб.: Питер, 2004.
2. Страуструп Б. Язык программирования C++. Специальное издание. — М.: Бином, 2011.
3. Липпман С., Лажойе Ж., Му Б. Язык программирования C++. Базовый курс. 5-е изд. — М.: Вильямс, 2014.

Дополнительная литература

4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2007.
5. Майерс С. Эффективный и современный C++. — М.: Вильямс, 2016.
6. Страуструп Б. Программирование: принципы и практика с использованием C++. 2-е изд. — М.: Вильямс, 2016.

Электронные ресурсы

7. Учебник по C++ (learncpp): www.learncpp.com
8. Справочник по C++ и стандартной библиотеке: en.cppreference.com

- Спасибо за внимание!